1. Hadoop生态整合

1.1. 整合概述

Kudu 除了支持高吞吐离线分析（类似HDFS）和高并发随机读写（类似HBase），还可以整合主流分布式计算框架进行离线运算和即系查询，常见整合方案如下：

整合方案	用途	说明
Kudu+Spark	利用 Spark 实现对 Kudu 上数据的批处理，SQL 支持好	Kudu 提供了 Spark 专用库，可直接把 Kudu 和 Spark 的 Dataset 和 Dataframe 集成使用，甚至直接 SQL 操作 Kudu，如有需要也可以 RDD 集成 Kudu。
Kudu+Flink	借助 Flink 实现对 Kudu上数据的批处理（读写）、流处理（写）	Flink 有非常强的批处理和流处理能力，流处理能力强于 Spark，依赖 bahir-flink 项目
Kudu+Impala	利用 Impala 基于 SQL 在线分析存储在 Kudu 上的数据	Impala 支持 SQL 直接操作 Kudu；同为 cloudera 公司 开发，整合更好；Impala 基于 MPP 架构
Kudu原生API	适合开发普通应用程序（非分析类）	没有SQL支持，开发门槛稍高

1.2. 集成 Spark

1.2.1. Spark shell 中操作 Kudu

在 Spark shell 中可以轻松操作 Kudu，不过这种方式不常用，参考链接如下：https://kudu.apache.org/releases/1.10.0/docs/developing.html#_kudu_integration_with_spark

1.2.2. 代码整合 Kudu+Spark

1.2.2.1. 项目准备

修改 pom.xml，最中内容如下：

```
<properties>
    <kudu.version>1.11.1</kudu.version>
    <junit.version>4.12</junit.version>
    <scala.version>2.11.8</scala.version>
    <spark.version>2.4.7</spark.version>
</properties>

<dependencies>
    <!-- Scala -->
    <dependency>
        <groupId>org.scala-lang</groupId>
        <artifactId>scala-library</artifactId>
        <version>${scala.version}</version>
    </dependency>

    <!-- Spark -->
    <dependency>
        <groupId>org.apache.spark</groupId>
        <artifactId>spark-core_2.11</artifactId>
        <version>${spark.version}</version>
    </dependency>
    <dependency>
        <groupId>org.apache.spark</groupId>
        <artifactId>spark-sql_2.11</artifactId>
        <version>${spark.version}</version>
    </dependency>
    <dependency>
        <groupId>org.apache.spark</groupId>
        <artifactId>spark-hive_2.11</artifactId>
        <version>${spark.version}</version>
    </dependency>

    <!-- kudu-spark -->
    <dependency>
        <groupId>org.apache.kudu</groupId>
        <artifactId>kudu-spark2_2.11</artifactId>
        <version>${kudu.version}</version>
    </dependency>
    <!-- Kudu client -->
    <dependency>
```

```
        <groupId>org.apache.kudu</groupId>
        <artifactId>kudu-client</artifactId>
        <version>${kudu.version}</version>
    </dependency>

    <!-- Log -->
    <dependency>
        <groupId>org.slf4j</groupId>
        <artifactId>slf4j-simple</artifactId>
        <version>1.7.12</version>
    </dependency>

    <!-- JUnit test -->
    <dependency>
        <groupId>junit</groupId>
        <artifactId>junit</artifactId>
        <version>${junit.version}</version>
        <scope>provided</scope>
    </dependency>
</dependencies>
```

1.2.2.2. 准备数据文件

在项目根目录创建目录 dataset，并把数据文件 students.txt 放进去

1.2.2.3. 准备代码骨架

按照正常普通的方式，创建一个maven项目即可。

1.2.2.4. 基本操作

首先，我们定义一个常量 KUDU_MASTERS 存放 Kudu master 的连接信息：

```
//master连接信息
val KUDU_MASTERS = "bigdata02:7051,bigdata03:7051,bigdata04:7051"
```

接下来，我们创建case class Student，它的字段信息必须跟我们前面数据文件students100k相匹配：

```
case class Student(sid: Int, name: String, gender: String, age: Int, height:
Float, weight: Float)
```

1.2.2.5. DDL操作

代码实现如下：

```
@Test
def ddl(): Unit = {
    // 1. SparkSession
    val spark = SparkSession.builder()
```

```
        .master("local[6]")
        .appName("KuduSparkDemo")
        .getOrCreate()

    //2 创建 KuduContext
    val kuduContext = new KuduContext(KUDU_MASTERS, spark.sparkContext)

    //3、判断表是否存在，如果存在则删除表
    val TABLE_NAME = "students"
    if (kuduContext.tableExists(TABLE_NAME)) {
        kuduContext.deleteTable(TABLE_NAME)
    }

    //4. 定义一张Kudu表：students
    //4.1 定义字段信息
    val schema = StructType(
        List(
            StructField("sid", IntegerType, nullable = false),
            StructField("name", StringType, nullable = false),
            StructField("gender", StringType, nullable = false),
            StructField("age", IntegerType, nullable = false),
            StructField("height", FloatType, nullable = false),
            StructField("weight", FloatType, nullable = false)
        )
    )

    //4.2 定义主键(rowkey)
    val keys = Seq("sid")

    //4.3 定义分区信息
    import  scala.collection.JavaConverters._

    val numBuckets = 6
    val options = new CreateTableOptions()
        .addHashPartitions(List("sid").asJava,numBuckets)
        .setNumReplicas(1)

    //5. 创建一张Kudu表：students
    kuduContext.createTable(tableName = TABLE_NAME, schema = schema, keys =
keys, options = options)

    //6、关闭资源
    spark.close()
}
```

1.2.2.6. CUD操作

代码实现如下：

```
@Test
def cud(): Unit = {

    // 1、SparkSession
    val spark = SparkSession.builder()
        .master("local[6]")
```

```
        .appName("KuduSparkDemo")
        .getOrCreate()

    //2、创建 KuduContext
    val kuduContext = new KuduContext(KUDU_MASTERS, spark.sparkContext)

    // 3. 增加
    import spark.implicits._
    val df = Seq(
        Student(8, "王荣", "F", 19, 164.4f, 116.5f),
        Student(9, "李晓", "F", 18, 174.4f, 126.5f)
    ).toDF()

    val TABLE_NAME = "students" kuduContext.insertRows(df,  TABLE_NAME)

    // 4. 删除
    kuduContext.deleteRows(df.select($"sid"),  TABLE_NAME)

    // 5. 增或改
    kuduContext.upsertRows(df,  TABLE_NAME)

    // 6. 修改
    kuduContext.updateRows(df,  TABLE_NAME)

    //7、关闭资源
    spark.close()
}
```

用如下命令以验证结果（换成自己的主机名或者IP）：

```
kudu table scan node01:7051,node02:7051,node03:7051 students
```

1.2.2.7. Spark批处理-写Kudu

批量写代码实现：

```
@Test
def batchWrite(): Unit = {

    // 1.SparkSession
    val spark = SparkSession.builder()
    .master("local[6]")
    .appName("KuduSparkDemo")
    .getOrCreate()

    // 2.定义数据schema
    val schema = StructType(
        List(
            StructField("sid", IntegerType, nullable = false),
            StructField("name", StringType, nullable = false),
            StructField("gender", StringType, nullable = false),
            StructField("age", IntegerType, nullable = false),
            StructField("height", FloatType, nullable = false),
            StructField("weight", FloatType, nullable = false)
        )
```

```
    )

    // 3.从csv读取数据
    val studentsDF = spark.read
    .option("header", value = true)
    .option("delimiter", value = "\t")
    .schema(schema)
    .csv("dataset/students100k")

    // 4.写入Kudu
    val TABLE_NAME = "students"

    studentsDF.write
    .option("kudu.table", TABLE_NAME)
    .option("kudu.master", KUDU_MASTERS)
    .mode(SaveMode.Append)
    .format("kudu")
    .save()

    //5.回收资源
    spark.close()
}
```

1.2.2.8. Spark 批处理-读 Kudu（SQL分析）

```
@Test
def batchRead(): Unit = {

    // 1.SparkSession
    val spark = SparkSession.builder()
    .master("local[6]")
    .appName("KuduSparkDemo")
    .getOrCreate()

    // 2.从kudu表读取数据到DataFrame
    val TABLE_NAME = "students"

    val studentsDF = spark.read
    .option("kudu.table", TABLE_NAME)
    .option("kudu.master", KUDU_MASTERS)
    .format("kudu")
    .load()

    // 3.直接使用Spark API查询
    //studentsDF.select("sid","name", "gender", "age").filter("sid >= 5 and
sid<=10").show()

    // 3.基于DataFrame创建临时视图(临时表)
studentsDF.createOrReplaceTempView("students")

    // 4.执行sql查询
    // val projectDF =
    // spark.sql("select sid, name, gender, age from students where age <= 19
and height > 180")
```

```
    val projectDF = spark.sql("select gender, count( ), max(height),
min(height), avg(height) from students where age <= 19 and height > 180 group by
gender")

    //5.打印结果
    projectDF.show()

    //6.关闭资源
    spark.close()
}
```

1.2.2.9. 特别注意

每个集群避免多KuduClient

常见错误就是创建了多个KuduClient对象。在kudu-spark中，KuduClient对象由KuduContext所持有。对于同一kudu集群，不应该创建多个KuduClient对象，而是应该通过KuduContext访问KuduClient，方法为KuduContext.syncClient。

存在问题和限制

Spark2.2+需要Java8支持，尽管Kudu Spark2.x兼容Java 7。Spark-2.2默认依赖Kudu-1.5.0。

Kudu表如果包含大写或非ascii字符的话，注册临时表时需要指定其他的名字。

列名如果含有大写或非ascii字符的，不能在Spark SQL中使用，否则必须重命名。

<>或or操作不会推送到kudu执行，而是最终由Spark的task来计算。只有以通配结尾的like运 算才会推送到kudu执行，比如like foo%，但是like foo%bar则不会推送给Kudu。并不是Spark SQL中的每种类型Kudu都支持，Date和Complex类型就不支持。

Kudu表在Spark SQL中只能注册成临时表，不能使用HiveContext访问。

1.3. 集成 Flink

1.3.1. 集成说明

在 Spark 和 Flink 先后崛起之后，开始与 Hadoop 生态中的各个组件整合（官方或者第三方）。Apache Bahir 就是一个第三方项目，它对 Spark 和 Flink 进行扩展以便于它们整合其他组件（主要针对流处理）。 以下是 Apache Bahir 的官网：http://bahir.apache.org/

What is Apache Bahir

Apache Bahir provides extensions to multiple distributed analytic platforms, extending their reach with a diversity of streaming connectors and SQL data sources.

Currently, Bahir provides extensions for Apache Spark and Apache Flink.

Apache Spark extensions

- Spark data source for Apache CouchDB/Cloudant
- Spark Structured Streaming data source for Akka
- Spark Structured Streaming data source for MQTT
- Spark DStream connector for Apache CouchDB/Cloudant
- Spark DStream connector for Akka
- Spark DStream connector for Google Cloud Pub/Sub
- Spark DStream connector for PubNub new
- Spark DStream connector for MQTT new (new Sink)
- Spark DStream connector for Twitter
- Spark DStream connector for ZeroMQ new (Enhanced Implementation)

Apache Flink extensions

- Flink streaming connector for ActiveMQ
- Flink streaming connector for Akka
- Flink streaming connector for Flume
- Flink streaming connector for InfluxDB new
- Flink streaming connector for Kudu new
- Flink streaming connector for Redis
- Flink streaming connector for Netty

The Apache Bahir community welcomes the proposal of new extensions.

Apache Bahir 对 Flink 的支持以子项目 bahir-flink 的方式提供，以下是它的 github 主页：
https://github.com/apache/bahir-flink

本节我们就基于 bahir-flink 来整合 Kudu+Flink，目前支持：

```
批处理读和写
流处理写（流处理读一般只针对消息队列，对于存储流处理读意义不大）
```

1.3.2. 编译 bahir-flink

关于版本：

```
bahir-flink目前1.0版还没正式发布，刚到1.0-rc5，且不支持Kudu。1.1版开始支持Kudu，目前还在1.1-SNAPSHOT版（Kudu1.10.0和Flink1.9.0），因此我们需要自己编译，且一定要在 Linux 或者 MacOS 下编译
```

编译并安装到maven本地仓库

```
git clone https://github.com/apache/bahir-flink.git
cd bahir-flink/
mvn -DskipTests -Drat.skip=true clean install
```

如果有些依赖包实在下载不下来导致编译不过的话，可以使用老师提供的编译好的包直接安装到 maven 本地仓库即可：

```
mvn install:install-file -DgroupId=org.apache.bahir -DartifactId=flink-
connector-kudu_2.11 -Dversion=1.1-SNAPSHOT -Dpackaging=jar -Dfile=./flink-
connector-kudu_2.11-1.1-SNAPSHOT.jar
```

1.3.3. 项目准备

修改 pom.xml，最终内容如下：

```
<properties>
    <kudu.version>1.11.1</kudu.version>
    <junit.version>4.12</junit.version>
    <scala.version>2.11.8</scala.version>
    <spark.version>2.4.5</spark.version>
    <flink.version>1.9.0</flink.version>
    <flink-connector.version>1.1-SNAPSHOT</flink-connector.version>
</properties>

<dependencies>
    <!-- bahir-flink -->
    <dependency>
        <groupId>org.apache.bahir</groupId>
        <artifactId>flink-connector-kudu_2.11</artifactId>
        <version>${flink-connector.version}</version>
    </dependency>

    <dependency>
        <groupId>org.apache.flink</groupId>
        <artifactId>flink-streaming-java_2.11</artifactId>
        <version>${flink.version}</version>
        <scope>provided</scope>
    </dependency>

    <!-- Scala -->
    <dependency>
        <groupId>org.scala-lang</groupId>
        <artifactId>scala-library</artifactId>
        <version>${scala.version}</version>
    </dependency>

    <!-- Spark -->
    <dependency>
        <groupId>org.apache.spark</groupId>
        <artifactId>spark-core_2.11</artifactId>
        <version>${spark.version}</version>
    </dependency>
    <dependency>
        <groupId>org.apache.spark</groupId>
        <artifactId>spark-sql_2.11</artifactId>
        <version>${spark.version}</version>
    </dependency>
    <dependency>
        <groupId>org.apache.spark</groupId>
        <artifactId>spark-hive_2.11</artifactId>
        <version>${spark.version}</version>
```

```
    </dependency>

    <!-- kudu-spark -->
    <dependency>
        <groupId>org.apache.kudu</groupId>
        <artifactId>kudu-spark2_2.11</artifactId>
        <version>${kudu.version}</version>
    </dependency>

    <!-- Kudu client -->
    <dependency>
        <groupId>org.apache.kudu</groupId>
        <artifactId>kudu-client</artifactId>
        <version>${kudu.version}</version>
    </dependency>

    <!-- Log -->
    <dependency>
        <groupId>org.slf4j</groupId>
        <artifactId>slf4j-simple</artifactId>
        <version>1.7.12</version>
    </dependency>

    <!-- JUnit test -->
    <dependency>
        <groupId>junit</groupId>
        <artifactId>junit</artifactId>
        <version>${junit.version}</version>
        <scope>provided</scope>
    </dependency>
</dependencies>
```

1.3.4. 批处理读写

1.3.4.1. 批处理读

批处理读需要开启 Kudu 安全，这里就不做演示了。批处理读使用 KuduInputFormat，代码如下：

```
@Test
public void testBatchRead() throws Exception {
    //1、初始化执行环境
    ExecutionEnvironment env = ExecutionEnvironment.getExecutionEnvironment();
env.setParallelism(3);

    //2、构建数据处理逻辑(输入-->处理--输出)
    //2.1 输入(读取kudu的students表)

    //a、创建KuduReaderConfig
    KuduReaderConfig kuduReaderConfig = KuduReaderConfig.Builder
        .setMasters(KUDU_MASTERS)
        .build();

    //b、创建KuduTableInfo
    KuduTableInfo tableInfo = KuduTableInfo.Builder
        .create("students").replicas(1).addColumn(KuduColumnInfo.Builder
```

```
        .create("sid",Type.INT32).key(true).hashKey(true).build())
        .addColumn(KuduColumnInfo.Builder.create("name",  Type.STRING).build())
        .addColumn(KuduColumnInfo.Builder.create("gender",
 Type.STRING).build())
        .addColumn(KuduColumnInfo.Builder.create("age",  Type.INT32).build())
        .addColumn(KuduColumnInfo.Builder.create("height",  Type.FLOAT).build())
        .addColumn(KuduColumnInfo.Builder.create("weight",  Type.FLOAT).build())
        .build();


    //c、创建反序列化器KuduDeserialization
    KuduDeserialization serDe = new PojoSerDe(Student.class);

    //d、组装过滤条件
    List<KuduFilterInfo> tableFilters = new ArrayList<>();



 tableFilters.add(KuduFilterInfo.Builder.create("age").greaterThan(18).build());
tableFilters.add(KuduFilterInfo.Builder.create("age").lessThan(20).build());

    //e、指定要返回的列
    List<String> tableProjections = Arrays.asList("sid", "age");

    //f、组装KuduInputFormat
    DataSet<Student> result = env.createInput(new
KuduInputFormat(kuduReaderConfig, tableInfo, serDe, new ArrayList<>(),
tableProjections),  TypeInformation.of(Student.class));

    //2.2 处理(包含输出)
    result.count();

    //3、执行job(延迟执行)
    env.execute();
}
```

1.3.4.2. 批处理写

```
@Test
public void testBatchWrite() throws Exception {
    //1、初始化执行环境
    ExecutionEnvironment env = ExecutionEnvironment.getExecutionEnvironment();
env.setParallelism(3);

    //2、构建数据处理逻辑(输入-->处理--输出)
    //2.1 输入
    DataSet<Student> originData= env.readCsvFile("dataset/students100k")
        .fieldDelimiter("\t")
        .ignoreFirstLine()
        .ignoreInvalidLines()
        .pojoType(Student.class,"sid","name","gender","age","height","weight");

    //2.2 处理(咱们不处理)

    //2.3 输出(kudu 表)
```

```
    //a、创建KuduWriterConfig
    KuduWriterConfig  writerConfig=KuduWriterConfig.Builder
        .setMasters(KUDU_MASTERS)
        .setWriteMode(KuduWriterMode.UPSERT)
        .setConsistency(SessionConfiguration.FlushMode.AUTO_FLUSH_BACKGROUND)
        .build();

    //b、创建KuduTableInfo
    KuduTableInfo tableInfo = KuduTableInfo.Builder
        .create("students1")
        .replicas(1)
        .addColumn(KuduColumnInfo.Builder.create("sid",
Type.INT32).key(true).hashKey(true).build())
        .addColumn(KuduColumnInfo.Builder.create("name",   Type.STRING).build())
        .addColumn(KuduColumnInfo.Builder.create("gender",
Type.STRING).build())
        .addColumn(KuduColumnInfo.Builder.create("age",   Type.INT32).build())
        .addColumn(KuduColumnInfo.Builder.create("height",
Type.FLOAT).build())
        .addColumn(KuduColumnInfo.Builder.create("weight",
Type.FLOAT).build())
        .build();

    //c、创建KuduSerialization
    KuduSerialization serDe=new PojoSerDe(Student.class);

    //d、装配KuduOutputFormat
    originData.output(new KuduOutputFormat(writerConfig, tableInfo, serDe));

    //3、执行job(延迟执行)
    env.execute();
}
```

1.3.4.3. 流处理写

直接使用 KuduSink：

```
@Test
public void testKuduSink() throws Exception {
    List<Student> list = Arrays.asList(
        new Student(1, "张三", "F", 19, 176.3f, 134.4f),
        new Student(2, "李四", "F", 20, 186.3f, 154.8f)
    );
    //1、初始化执行环境StreamExecutionEnvironment env =
    StreamExecutionEnvironment.getExecutionEnvironment(); env.setParallelism(3);

    //2、构建数据处理逻辑(输入-->处理--输出)
    //2.1 输入
    DataStream<Student> originData = env.fromCollection(list);
    //2.2 处理(咱们不处理)
    //2.3 输出(kudu 表)
    //a、创建KuduWriterConfig
    KuduWriterConfig  writerConfig=KuduWriterConfig.Builder
        .setMasters(KUDU_MASTERS)
        .setWriteMode(KuduWriterMode.UPSERT)
```

```
        .setConsistency(SessionConfiguration.FlushMode.AUTO_FLUSH_BACKGROUND)
        .build();

    //b、创建KuduTableInfo
    KuduTableInfo tableInfo = KuduTableInfo.Builder
        .create("students1")
        .replicas(1)
        .addColumn(KuduColumnInfo.Builder.create("sid",
Type.INT32).key(true).hashKey(true).build())
        .addColumn(KuduColumnInfo.Builder.create("name",   Type.STRING).build())
        .addColumn(KuduColumnInfo.Builder.create("gender",
Type.STRING).build())
        .addColumn(KuduColumnInfo.Builder.create("age",   Type.INT32).build())

        .addColumn(KuduColumnInfo.Builder.create("height",
Type.FLOAT).build())
        .addColumn(KuduColumnInfo.Builder.create("weight",
Type.FLOAT).build())
        .build();
    //c、创建KuduSerialization
    KuduSerialization serDe=new PojoSerDe(Student.class);
    //d、装配KuduSink
    originData.addSink(new  KuduSink(writerConfig,tableInfo,serDe));

    //3、执行job(延迟执行)
    env.execute();
}
```